Software Design Document

Data Analysis and Visualization Tool

s5273814 – Nathanael Gazzard

s5309988 - Stephen Urquhart

s5318167 - Jahanzaib

Table of Contents

[1.0 System Vision 3](#_Toc144656117)

[1.1 Problem Background 3](#_Toc144656118)

[1.2 System Overview 3](#_Toc144656119)

[1.3 Potential Benefits 3](#_Toc144656120)

[2.0 Requirements 4](#_Toc144656121)

[2.1 User Requirements 4](#_Toc144656122)

[2.2 Software Requirements 5](#_Toc144656123)

[2.3 Use Cases & Use Case Diagrams 6](#_Toc144656124)

[2.3.1 Use Cases 6](#_Toc144656125)

[2.3.2 Case Diagrams 7](#_Toc144656126)

[3.0 Software Design and System Components 8](#_Toc144656127)

[3.1 Software Design 8](#_Toc144656128)

[3.2 System Components 9](#_Toc144656129)

[3.2.1 Functions 9](#_Toc144656130)

[3.2.2 Data Structures / Data Sources 11](#_Toc144656131)

[3.2.3 Detailed Design 13](#_Toc144656132)

[4.0 User Interface Design 14](#_Toc144656133)

[4.1 Visual Design 14](#_Toc144656134)

[Elements: 14](#_Toc144656135)

[Wireframe: 15](#_Toc144656136)

[Navigation (PC): 16](#_Toc144656137)

[Navigation (Mobile): 17](#_Toc144656138)

[Colour Palette: 18](#_Toc144656139)

[Graphic Sample: 19](#_Toc144656140)

[Font: 19](#_Toc144656141)

# System Vision

## Problem Background

Kaggle.com hosts a dataset of ~400k restaurant inspections of New York City (NYC) food establishments. The inspections span from January 2010 to August 2017 and are graded on an A-F scale, having been collected by the NYC Department of Health. A Data Analysis and Visualization Tool (DAVT) must be developed that allow a user to make the following five queries:

1. Retrieve all inspection details for a specified period.
2. Plot violations distributed on a per-suburb basis.
3. Retrieve all violations containing a specified keyword for a specified period.
4. Map violations pertaining to animals distributed over time and suburbs.
5. Pull the 100 places with the best improvement over the last year for each boro.

## System Overview

The system will have three distinct views:

1. The home page from which the type of query is selected, and refining details specified.
2. The data result page which lists the results of the selected query.
3. The data visualisation page which visualises the results of the data result page.

The home page can be returned to from either of the other pages to commence a new query instantly. The system will use clean minimal graphics, keeping the functionality foremost.

## Potential Benefits

The system will make accessing the desired data a straightforward process, with no distractions along the way.

It will allow for quickly switching back and forth between a raw data view and data visualisation to make comparisons.

It will make analysing the immense amount of data accessible without overwhelming the user.

# Requirements

## User Requirements

**Overview:**

This Data Analysis and Visualisation Tool is designed to empower restaurant owners and health inspectors in New York with a user-friendly solution to analyse and visualise restaurant inspection results. As a health inspector, Mary Smith, you need a seamless understanding of inspection data trends and make informed decisions to ensure public health safety.

**User Persona:**

Mary Smith is a dedicated health inspector responsible for conducting restaurant inspections across New York neighbourhoods. Her role requires her to analyse inspection data, identify trends, and take necessary actions to maintain food safety standards.

**Interaction and Usage:**

1. *Intuitive Dashboard:* Upon opening the software, Mary encounters an intuitive dashboard showcasing key metrics and visualisations of inspection data. This dashboard provides a platform that is easy to navigate and supports a linear workflow.
2. *Flexible Period Selection:* Mary can easily select a specific period for analysis. The tool allows her to choose a date range, enabling her to focus on recent or historical data.
3. *Violation Distribution Plot:* Mary navigates to the "Violation Distribution" section. Here, she selects a date range and instantly views a graphical representation of violation distribution across different suburbs. This helps her identify areas with higher instances of violations.
4. *Keyword-Based Violation Search:* In the "Keyword Search" feature, Mary enters keywords like "rodent" or "contamination." The tool displays a list of violations matching her search, aiding her in targeted investigation.
5. *Animal-Related Analysis:* Mary accesses the "Animal Analysis" section to analyse cases related to rodents, pests, and other animals. She can view trends over time and explore how animal-related violations are distributed across neighbourhoods.
6. *User-Friendly Export:* Mary accesses the “Status Improvement” feature, which displays the top 100 premises with the most improvement over twelve months per neighbourhood. She can use this data to analyse trends and the long-term results of investigations.

**User Needs:**

* Easily select specific periods for analysis.
* Visualise violation distribution over neighbourhoods.
* Search for violations based on keywords.
* Analyse trends related to animal-related violations.

The Data Analysis and Visualization Tool caters to the needs of health inspectors like Mary Smith, enabling them to make data-driven decisions to enhance food safety standards across New York's restaurants.

## Software Requirements

**R1.1** The tool shall provide a user-friendly graphical user interface (GUI) for easy interaction and navigation.

**R1.2** The GUI shall include a dashboard displaying key metrics and visualisations of inspection data.

**R1.3** Users shall be able to select a specific period for analysis using date range selectors.

**R2.1** The tool shall offer a "Violation Distribution" feature that generates a graphical plot showcasing the distribution of violations across different suburbs for the selected period.

**R2.2** A "Keyword Search" functionality shall allow users to enter keywords and retrieve a list of violations containing those keywords for the selected period.

**R2.3** An "Animal Analysis" section shall enable users to analyse trends related to animal-related violations (e.g., rodents, pests) over time and their distribution across neighbourhoods.

**R3.1** The system shall utilise a version control system (VCS) such as GitHub to track changes in the source code and collaborate on development.

**R4.1** The tool's GUI shall have a responsive design, ensuring usability on different screen sizes and devices.

**R4.2** User preferences, such as selected date ranges and visualisation settings, shall be saved and restored upon subsequent logins.

**R5.1** The tool shall implement appropriate security measures to safeguard user data and prevent unauthorised access.

**R5.2** The backend APIs shall efficiently handle data retrieval, processing, and storage for various analysis tasks.

**R5.3** The database shall store inspection data, user preferences, and other relevant information securely and efficiently.

**R5.4** The tool shall support concurrent usage by multiple users without compromising performance.

**R5.5** The backend shall be scalable to accommodate a growing number of users and increasing data volumes.

**R6.1** The system shall maintain a detailed audit trail of user interactions and system activities for accountability and troubleshooting.

**R6.2** The tool shall ensure the accuracy and consistency of data used for analysis and visualisation tasks.

**R6.3** The GUI shall provide clear and intuitive labels, tooltips, and error messages to guide users during interaction.

## Use Cases & Use Case Diagrams

### Use Cases

**Restaurant Owner - Trend Analysis**

Actor: Sarah, a restaurant owner.

Scenario: Sarah wants to track the inspection trends for her restaurant over the past year.

Steps:

1. Sarah opens the tool.
2. She selects the date range from the past year.
3. Sarah uses the "Violation Distribution" feature through the query field.
4. The tool generates a plot showing the distribution of violations in different suburbs for the selected period.
5. Sarah notices increased violations in a specific suburb and decides to take corrective actions.

**Health Inspector – Keyword Search**

Actor: John, a health inspector.

Scenario: John needs to investigate cases related to “rodents” reported in the last month.

Steps:

1. John accesses the tool’s “Keyword Search” feature through the query field.
2. He enters the keyword “rodents” and selects the last month as the date range.
3. The tool retrieves and displays a list of violations containing the keyword “rodents.”
4. John identifies patterns in the violations and decides to conduct targeted inspections in the affected areas.

**Analyst - Animal-Related Analysis**

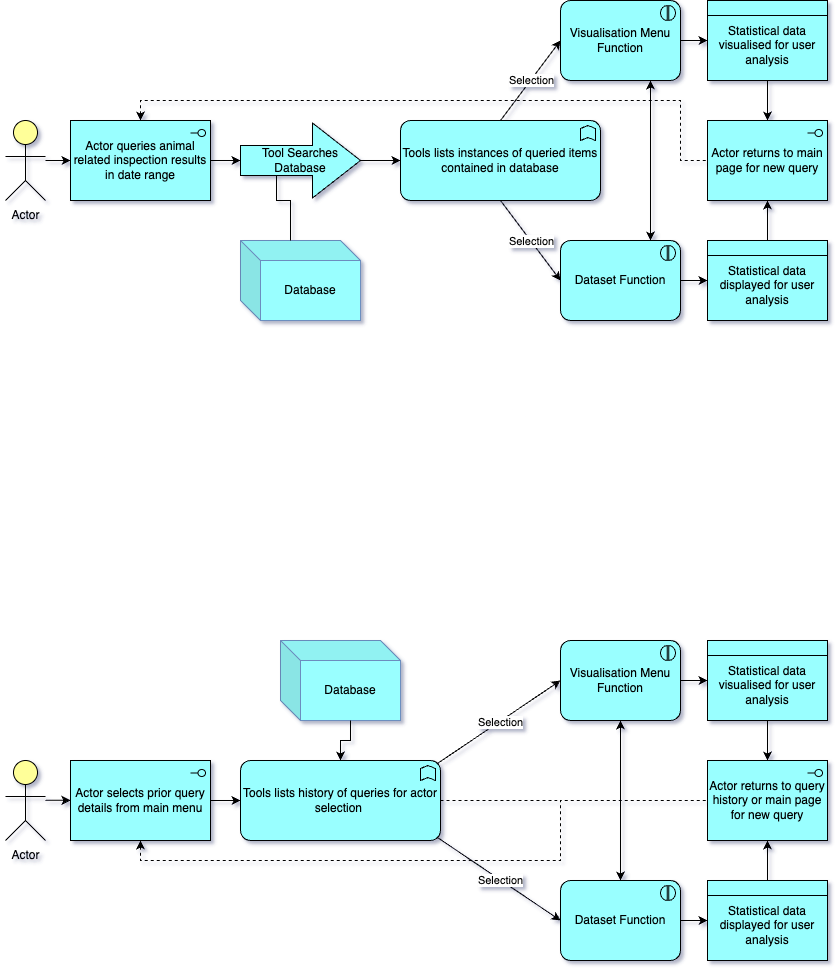
Actor: Lisa, a data analyst.

Scenario: Lisa is tasked with analysing the trend of animal-related violations across neighbourhoods.

Steps:

1. Lisa opens the tool.
2. She goes to the "Animal Analysis" section.
3. Lisa selects the past two years as the date range.
4. The tool presents visualisations depicting the trend of animal-related violations over time.
5. Lisa identifies neighbourhoods with consistent animal-related issues and suggests targeted awareness campaigns.

### Case Diagrams



# Software Design and System Components

## A diagram of a workflow Description automatically generatedSoftware Design

## System Components

### Functions

**Retrieve Inspection Details**

*Description*: Retrieves relevant inspection details for a user-selected period.

*Input Parameters*:

* + - Start date (date)
    - End date (date)

*Side Effects*: None

*Return Value*: List of inspection details.

**Plot Violation Distribution**

*Description*: Generates a plot showing the distribution of violations over different suburbs for a user-selected period.

*Input Parameters*:

* + - Start date (date)
    - End date (date)

*Side Effects*: None

*Return Value*: Plot or visualisation.

**Retrieve Violations by Keyword**

*Description*: Retrieves all violations that contain a user-entered keyword for a user-selected period.

*Input Parameters*:

* + - Keyword (string)
    - Start date (date)
    - End date (date)

*Side Effects*: None

*Return Value*: List of violations matching the keyword.

**Analyse Animal-Related Cases**

*Description*: Analyses cases related to animals like rats, mice and their trends over time and distribution over suburbs.

*Input Parameters*:

* + - Start date (date)
    - End date (date)

*Side Effects*: None

*Return Value*: Statistical analysis results, e.g., trends, distribution.

**Additional Analysis Tool: Temporal Analysis of Inspection Trends**

*Description*: Implement a tool that allows users to perform a temporal analysis of restaurant inspections over time. This tool would help users understand how inspection trends have evolved over different months or years.

*Input Parameters*:

* Start date (date)
* End date (date)
* Granularity (e.g., month, year)

*Side Effects*: None

*Return Value*: This tool could produce various insights, including:

* A line chart or bar chart showing the number of inspections conducted over the selected time period, aggregated by the chosen granularity (e.g., monthly or yearly).
* Highlighting any seasonal trends in inspection frequency.
* Identifying periods with significant changes in inspection frequency.

### Data Structures / Data Sources

**Data Source: New York Restaurant Inspection Dataset**

*Type of Structure*: External dataset (CSV, JSON, or database)

*Description*: This is the primary data source containing restaurant inspection records for New York City.

*Data Members*: Various columns in the dataset (attributes like restaurant name, inspection date, violation description, borough, etc.).

*Functions That Use It*: All functions related to data retrieval and pre-processing will utilize this dataset.

**Data Structure: Pandas DataFrame.**

*Type of Structure*: DataFrame (from the Pandas library)

*Description*: DataFrames are used to efficiently store and manipulate tabular data from the external dataset.

*Data Members*: Columns represent different attributes, and rows represent individual inspection records.

*Functions That Use It*: All data retrieval, filtering, and preliminary analysis functions will use Pandas DataFrames.

**Data Structure: Python Lists and Numpy Arrays**

*Type of Structure*: Lists (built-in) and Numpy arrays

*Description*: Lists and arrays are used for storing and processing data elements, such as counts of violations or aggregated results.

*Data Members*: Elements in the list or array can represent counts, aggregated statistics, or subsets of data.

*Functions That Use It*: Functions related to data analysis, statistical calculations, and intermediate storage will use lists and Numpy arrays.

**Data Structure: Python Dictionaries**

*Type of Structure*: Dictionaries (built-in)

*Description*: Dictionaries are used to store key-value pairs for various purposes, such as mapping borough names to inspection counts.

*Data Members*: Key-value pairs where keys represent unique identifiers, and values store related information.

*Functions That Use It*: Functions related to data aggregation, mapping information, and result storage will use Python dictionaries.

**Data Structure: Matplotlib and Seaborn Plots/Charts**

*Type of Structure*: Visual representations (Matplotlib and Seaborn)

*Description*: Visual plots and charts are used to present data visually to users, such as bar charts, line charts, and scatter plots.

*Data Members*: Various elements required for creating and displaying visualizations, including data points, axis labels, and chart types.

*Functions That Use It*: Functions responsible for data visualization will use Matplotlib and Seaborn to render charts and plots.

With Python as our chosen programming language, Pandas, NumPy, and Matplotlib are excellent libraries for data manipulation, analysis, and visualization. we can leverage these tools effectively to build our data analysis and visualization tool for New York Restaurant Inspection data.

### Detailed Design

**Pseudocode for Plotting Violation Distribution:**

# Function to plot violation distribution

function plot\_violation\_distribution(start\_date, end\_date):

# Filter data frame to include inspections within the specified date range

filtered\_data = dataset[(dataset['inspection\_date'] >= start\_date) & (dataset['inspection\_date'] <= end\_date)]

# Group data by suburb and count violations

violation\_counts = filtered\_data.groupby('borough')['violation\_description'].count()

# Create a bar chart to visualize violation distribution

create\_bar\_chart(violation\_counts)

**Pseudocode for Retrieving Violations by Keyword:**

# Function to retrieve violations by keyword

function retrieve\_violations\_by\_keyword(keyword, start\_date, end\_date):

# Filter data frame to include inspections within the specified date range

filtered\_data = dataset[(dataset['inspection\_date'] >= start\_date) & (dataset['inspection\_date'] <= end\_date)]

# Filter data to include rows containing the keyword

keyword\_matched\_data = filtered\_data[filtered\_data['violation\_description'].str.contains(keyword)]

# Return the filtered data as a Pandas DataFrame

return keyword\_matched\_data

**Pseudocode for Analyzing Animal-Related Cases:**

# Function to analyze animal-related cases

function analyze\_animal\_related\_cases(start\_date, end\_date):

# Filter data frame to include inspections within the specified date range

filtered\_data = dataset[(dataset['inspection\_date'] >= start\_date) & (dataset['inspection\_date'] <= end\_date)]

# Filter data to include rows related to animals (e.g., rats, mice)

animal\_related\_data = filtered\_data[filtered\_data['violation\_description'].str.contains('rat|mice|animal')]

# Perform statistical analysis on animal-related data

analysis\_results = perform\_statistical\_analysis(animal\_related\_data)

# Return the analysis results

return analysis\_results

These pseudocode examples illustrate how we might approach implementing the key functionalities of your data analysis software.

# User Interface Design

Group discussion led to the consensus that this system would be best served by an aesthetically clean and minimal design that would make the interface and data easy to parse. Initial sketches were created on paper to figure out the placement of each item before the final wireframes were created in draw.io.

## Visual Design

### Elements:

The title of the system will be displayed as a consistent header on all pages. \*

The “current query >” button will only appear if the user has previously made a query. When pressed, it will open the data page with the last query that was run.

The “query” field has a drop-down menu from which the user can select one of the five predefined queries. Upon doing so, the form beneath will update to allow the user to enter the contextually appropriate query parameters.

The "prior search details" link in the query details form will open a floating list of date-time stamps representing the user's prior query parameters for the current query type. The user can select one that will fill the form with the former values.

Pressing the submit button will run the query and take the user to the data results page provided the required parameters have been supplied.

The data and visualisation view can be toggled like browser tabs.

When a query or visualisation takes up more room than is available on screen, the display area will be scrollable.

The new query button on the data or visualisation page will take the user back to the home (query) page. The query field and parameters form will be prefilled with the details of the query they were just viewing.

\*Titles and names are subject to change and should be considered placeholders.

### Wireframe:

A group of blueprints

Description automatically generated

### Navigation (PC):

A screenshot of a computer screen

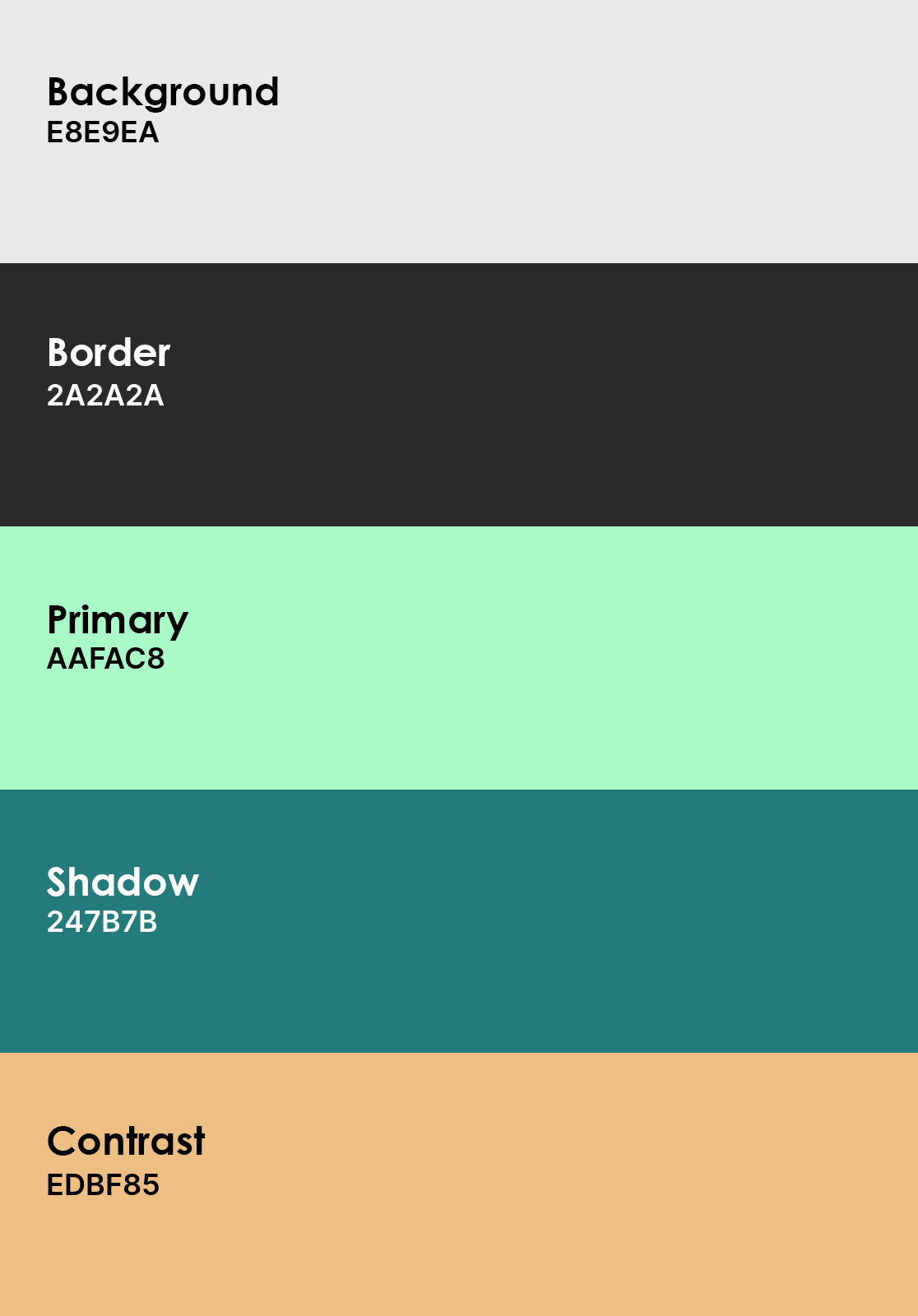
Description automatically generated

### Navigation (Mobile):

A screenshot of a computer

Description automatically generated

### Colour Palette:



### Graphic Sample:

A green rectangular sign with black text

Description automatically generated

### Font:

In line with the goal of clean design in mind, research was done into what font is the easiest to parse in a digital format. Sans-serif fonts were ubiquitous in their domination of this space, but foremost among them was Helvetica. However, a recurrent trend in the reading was Roboto as a suitable alternative. While Roboto is technically less minimalist than Helvetica, its liberties are in service of readability. Additionally, with the prospect of this system being utilised in mobile spaces, particularly Android devices, Roboto becomes a direct continuation of the native device experience. For these reasons, Roboto was selected over Helvetica.

Sample of the selected font, Roboto.